AMATH 482 Homework 1

Wenxuan Liu

January 27, 2021

Abstract

We are given 49 noisy 3-D acoustic data collected chronologically. We use fast Fourier transformation to get the realizations in frequency space. Then we average all 49 realizations together to determine the acoustic frequency on which we want to focus. Then we apply three Gaussian filters centered at the frequency to filter the data in three dimensions. We locate the data points with high amplitude and average their location to get a center points. We connect the center points to get the path of the submarine.

1 Introduction and Overview

We are hunting for a submarine in the Puget Sound using noisy acoustic data. The submarine emits some acoustic frequency, but we don't know the frequency and the data is noisy. Thus, we want to first know the frequency of the emitted sound, and then use this frequency to filter the data to get rid of white noise.

In order to get the frequency, we take advantage that we have 49 set of data. We take Fourier transformation on 49 data and average over them. The noise is canceled, so we can tell which frequency is emitted by the submarine. After doing Fourier transformation on each 3-D data, We apply a 3-D Gaussian filter centered at the above frequency to clean the data. Then we can average the positions with high amplitude, which is where the submarine is at. Repeating this on 49 3-D data to get 49 points, we get the path of submarine.

2 Theoretical Background

2.1 Fourier Transformation

As we learned from our text book (lecture 1) [1], Fourier introduced the concept of representing a given function $f(x)$ by a trigonometric series of sines and cosines:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_n \cos kx + b_n \sin kx) \quad x \in [-\pi, \pi]. \tag{1}$$

Based on this idea, we have Fourier transformation that can bring any function from time/spacial domain to frequency domain:

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx} dx \tag{2}$$

The inverse Fourier transformation is the inverse transformation of Fourier transformation, which brings the function back from frequency domain:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(k) e^{ikx} dk \tag{3}$$

For a sequence of N values, we have discrete Fourier Transform:

$$\hat{x} = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi i k n}{N}} \tag{4}$$

Fast Fourier Transform is an algorithm that run in O(N log(N)). This algorithm allow us to do discrete Fourier Transform (Equation 4) fast. MATLAB has built-in function that perform FFT on data sets.

2.2 Gaussian filter

From text book (lecture 2) [1], we learned Gaussian filters, which has the form:

$$F(k) = e^{-\tau(k-k_0)^2} \tag{5}$$

By applying filter in frequency space, we can preserve the frequency we want and git rid of the noisy data. After transform the filtered frequency data back to time/spacial domain, the data is less noisy.

2.3 Averaging over realizations

From text book (lecture 3) [1], one way of denoise the data is to average over realizations in frequency domain. If we do FFT on each shifted data set and add sum up all frequency data, the random noise cancel with each other. Thus, the frequency we want will stand out.

3 Algorithm Implementation and Development

We start by loading the subdata, which is a 262144×49 data. The data contain acoustic data of a spacial domain [-10, 10] in three directions. We process the data in steps described below:

1. First we prepare/interpret the data and define some helper variables. Since FFT takes spacial data in domain $[-\pi, \pi]$, we scale the coordinate data by $\frac{2\pi}{2L}$, $L = 10$. There are $64 \times 64 \times 64$ data points collected in the 3-dimension space. Such spacial data are collected 49 times (every 0.5h) during the day. We can define n=64, realization=49. We also define x,y,x and k to form spacial and frequency spaces, and create mesh-grids with them.

2. Next, through averaging of the spectrum, we can determine the frequency signature generated by the submarine. We first get the 3D data from 262144 data points by calling reshape(). Next, we call fftn() on each spacial data, and then average all 49 shifted data in frequency domain. To get average, we need to take absolute value of our data as they are complex numbers; we also need to fftshift() in order to get the correct ordering. The ones with highest amplitude is the frequency we want, so we use max() and ind2sub() to get the index of maximum values in each dimension. We record three of them for later use. See Algorithm 1 for sudo-code.

Algorithm 1: Averaging Algorithm

```
Define Untave
for j = 1 : 49 do
    Reshape the j^th column of subdata into 64^3
    Untave = Untave + fftn(reshaped data)
end for
Untave = abs(fftshift(Untave)/49)
Get max value and index of Untave
Get 3D index by ind2sub()
```

3. Now we get the frequency signatures we want, so we can filter out other frequencies. We need to make filters that center at the frequencies we got with proper width. To do this, we just create three Gaussian functions centered at the three frequencies. The three Gaussian filters are multiplied to work in 3D space. The last step is to do fftshift on our filter because it is going to be applied to the frequency data after doing fftn.

4. With the Gaussian filter and the spacial data, we can start looking for the center location of the signal. For each spacial data, we need to reshape() the data, and then call fftn() on it to get frequency data. Multiply the frequency data with 3D filter, we get only the submarine frequency. Transform the

frequency data back to spacial domain. Compute the index of maximum amplitude then get and store actual coordinate value from indices by multiplying $\frac{20}{64}$ and minus 10. See Algorithm 2 for sudo-code.

Algorithm 2: Filtering Algorithm

```
Define centers
for j = 1 : 49 do
    Reshape the j^th column of subdata into 64^3
    Utn = fftn(Un)
    Unft = filter_xyz.*Utn
    Unf = ifftn(Unft)
    Get max value and index
    Get 3D index by ind2sub()
end for
```

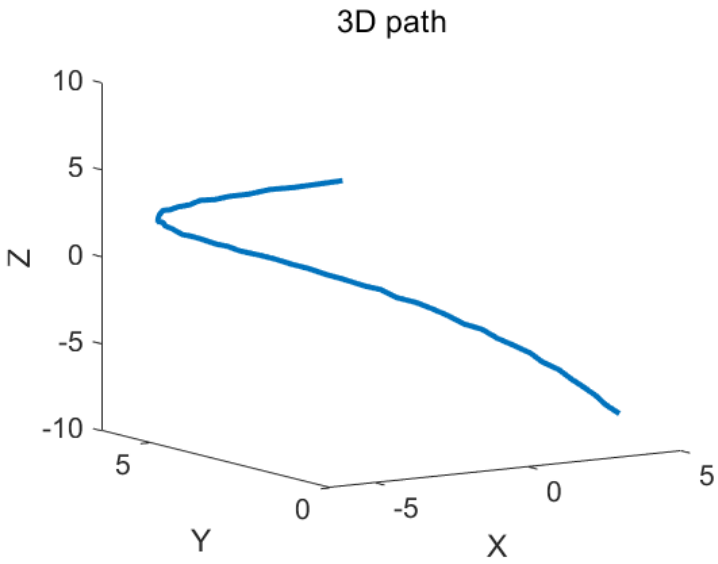
5. In the end, we put the x and y coordinates in a table and plot 2D and 3D plots to follow the submarine.

4 Computational Results

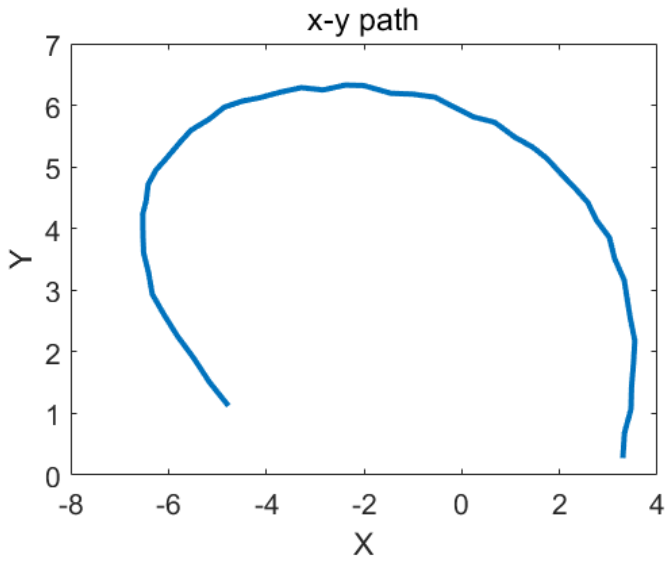
After the above processes, we get the actual x,y,z position of the submarine at 49 moments. We only need the x and y coordinates to send our air force. See Table 1 for coordinate results. See Figure 1 for the 2D and 3D plots of the submarine path.

X(1:24)	Y	Z		X(25:49)	Y	Z
3.2947	0.27525	-7.6904		-2.0083	6.3208	-0.49375
3.3275	0.6848	-7.392		-2.3838	6.3281	-0.16959
3.4604	1.069	-7.0941		-2.8335	6.25	0.074314
3.4736	1.4456	-6.7645		-3.2843	6.2863	0.42742
3.5116	1.8049	-6.5184		-3.7081	6.2138	0.71265
3.539	2.1795	-6.1704		-4.1328	6.123	0.94727
3.4489	2.5305	-5.9337		-4.5032	6.0657	1.3562
3.3902	2.8187	-5.5572		-4.8611	5.9681	1.6544
3.322	3.1629	-5.3295		-5.166	5.7754	2.0117
3.125	3.5146	-5.0119		-5.5391	5.5957	2.2188
3.0195	3.8555	-4.6461		-5.7649	5.3997	2.5472
2.7537	4.1356	-4.4359		-6.0449	5.1361	2.8892
2.5797	4.4211	-4.0772		-6.2562	4.9486	3.0777
2.3156	4.6564	-3.7422		-6.4158	4.7161	3.4737
2.0533	4.8696	-3.4509		-6.4555	4.4563	3.6665
1.7339	5.1431	-3.252		-6.5262	4.2339	3.9355
1.4441	5.3241	-2.8416		-6.5211	3.8825	4.2529
1.0938	5.4784	-2.6466		-6.5069	3.6005	4.6511
0.66993	5.7251	-2.2978		-6.4063	3.2813	4.8438
0.23585	5.8117	-2.0067		-6.3306	2.9344	5.1965
-0.14824	5.9675	-1.6767		-6.0938	2.6068	5.447
-0.54688	6.1318	-1.4223		-5.8185	2.2582	5.8426
-1.0078	6.1816	-1.0762		-5.4725	1.8863	6.09
-1.4474	6.1945	-0.80181		-5.1694	1.5161	6.4052
				-4.7729	1.1238	6.7469

Table 1: Spacial Coordinate data



(a) 3D plot



(b) x-y plot

Figure 1: position of submarine in 3D and x-y plane

5 Summary and Conclusions

After a series of computing, we can get a smooth path of the submarine's movement. We now know that the submarine is at (-4.7729, 1.1238), so we can send the P-8 Poseidon subtracking aircraft to this spot and complete the hunting.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y] = meshgrid(x,y)` returns 2-D grid coordinates based on the coordinates contained in the vectors `x` and `y`. X is a matrix where each row is a copy of `x`, and `Y` is a matrix where each column is a copy of `y`. The grid represented by the coordinates `X` and `Y` has `length(y)` rows and `length(x)` columns.
- `Y = reshape(X, sz)` reshapes inputted elements into a $n \times n \times n$ 3D array.
- `Y = fftn(X)` returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm.
- `X = ifftn(Y)` returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm.
- `M = max(A)` returns the maximum elements of an array.
- `[row,col] = ind2sub(sz,ind)` returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz. Here sz is a vector with two elements, where sz(1) specifies the number of rows and sz(2) specifies the number of columns.
- `plot3(X,Y,Z)` plots coordinates in 3-D space.
- `plot(X,Y)` creates a 2-D line plot of the data in Y versus the corresponding values in X.
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.

Appendix B MATLAB Code

MATLAB code that produce the outcomes discussed in this paper.

```
% Clean workspace
clear all; close all; clc

load subdata.mat % Imports the data as the 262144x49 (space by time) matrix called subdata

L = 10; % spatial domain
n = 64; % Fourier modes
x2 = linspace(-L,L,n+1);
x = x2(1:n); y = x;
z = x;

k = (2*pi/(2*L))*[0:(n/2 - 1) -n/2:-1];
ks = fftshift(k);

[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

% Average over all 49 data
Untave = zeros(1, n);
for j=1:49
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    Untave = Untave + fftn(Un);
end
% Calculate the index of the max frequency in Untave
Untave = abs(fftshift(Untave)/49);
[max_value,idx] = max(Untave(:));
[k0y,k0x,k0z] = ind2sub(size(Untave),idx);
```

```
% Use the index of max to find corresponding frequency(k) in x, y, z.
k0x = ks(k0x); k0y = ks(k0y); k0z = ks(k0z);
tau = 0.5;
% Creating Gaussian filters for the three frequencies.
filter_x = exp(-tau*(Kx - k0x).^2); % Define the filter for x
filter_y = exp(-tau*(Ky - k0y).^2); % Define the filter for y
filter_z = exp(-tau*(Kz - k0z).^2); % Define the filter for z
filter_xyz = fftshift(filter_x.*filter_y.*filter_z);
% Create containers of bubble centers.
centers = zeros(49, 3);
% Loop through all realizations to compute the exact submarine positions.
for j=1:49
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    % Filter using Gaussion filters.
    Utn = fftn(Un);
    Unft = filter_xyz.*Utn;
    Unf = ifftn(Unft);
    M = max(abs(Unf),[],'all');
    % Get positions with high amplitude.
    [bigy,bigx,bigz] = ind2sub(size(Unf),find(abs(Unf) > M*0.7));
    % Get average and call it the position of submarine.
    centers(j, 1) = mean(bigx)/64*20-10;
    centers(j, 2) = mean(bigy)/64*20-10;
    centers(j, 3) = mean(bigz)/64*20-10;
end
% Plot 3-D path
plot3(centers(:, 1), centers(:, 2), centers(:, 3));
xlabel('X');ylabel('Y');zlabel('Z')
figure();
% Plot just x-y plane path
plot(centers(:, 1), centers(:, 2));
xlabel('X');ylabel('Y')
```